Supplement to (84051) RK-WP Date-Handling Functions

Since submitting the above paper I have received & read Computers in Crisis by J. T. and M. J. Murray. (It can be ordered from the publishers directly--see the price & address in the review attached to my WP.) I attach many relevant pages from this important work. (Over half the book is pseudocode and (IBM) ALC listings of subroutines that sites can call to perform the date-handling functions described by the authors.)

I list below my points of agreement and disagreement with the authors.

1. **DISAGREE: They suggest calling the YEAR-DAY format "Neojulian".** (I think "Long Julian" would be better.) But preferably we should avoid all such idiosyncratic and non-parallel names (like their "commercial", "FIPS", and "European") and use instead names whose length and content contain useful information, such as MO-DA-YR, YR-MO-DA, and DA-MO-YR, and use YEAR-DAY rather than "Neojulian".

2. **AGREE / DISAGREE: They suggest storing YEAR-DAY in a 4-byte packed field,** rather than requiring DISPLAY. They claim that because some sites store 6-digit dates in packed 4-byte format, 4 bytes is the maximum that a replacement format should take. This point may be sound, but since some machines (e.g. micros) lack the ability to work with packed fields, it would seem better to use binary. The righthand two bytes would be the day, and the lefthand pair the 4-digit year.

3. **DISAGREE: They believe that data in external storage must be converted to carry a 4-digit year,** to enable valid date handling across the turn of the century. This is the basis for their difficult-to-do conversion plan (aptly described in "Chapter 11"). The simpler method is to convert only programs (to use date functions) and to let those functions contain a Virtual End Of the Century parameter (VEOC). In this way the worst impact of the turn of the century problem may be "dodged". The "mapping" (done within the function) of a virtual century (whose first year, say, is 03 & whose End year is 02) to the digits 00-99 can be accomplished in a half-dozen instructions or less. (Subtract the End year value + 1 from all years & add 100 to negative results.) (I now believe that specifying the end-year of a century is clearer than the method of my WP, which suggested specifying the Turn or start-year (00).)

One difficulty with converting files is that all programs that use those files must be identified, frozen, and switched over first. With the date-function dodge, the

converted one at a time. R. L. Conner agrees with me that this is a more practical solution for the real world (of conversion-allergic managers) than the method he personally prefers (converting files to an absolute date format). The penalty of "dodging" is merely the processing time taken in mapping virtually high years like 03 to really high years like 99. This is minor compared to a file conversion effort. And the processing penalty is temporary--once the turn of the century is out of the way, sites can drop the VEOC parameter and continue on their merry way. (Sites that want to convert to a more sensible date format can still do so, of course, and make use of the functions I've suggested as well.)

4. **AGREE: They suggest starting what I call the "absolute" date at Jan. 1, 1601, rather than at the astronomer's traditional Jan. 1, 4713 B.C.** This may simplify and speed some computations, be mentally simpler for coders to grasp, and will enable the date to be stored (as BINARY) in 3 bytes rather than 4. This will allow 2700 years or so before the absolute date wraps around to zero. (Probably 4 bytes should be what the function expects, though.) The division of such dates by 7 still yields modulus values that match Cobol's day-of-week conventions (1=Monday, etc.). (See p. 130 of the book.)

5. **DISAGREE: They suggest an upper limit on YEAR of 3400 AD,** since 3400 should not be a leap year (it will get us 1 day out of synch with the sun) although the current formula says that it should be. I assume the ISO will have issued a standard correcting the matter by then. (Maybe they could be requested to do so now, to enable stable universal date routines to be written. Meanwhile, it could just be assumed that they will do so.)

6. **AGREE: They provide return codes for all their "functions";** these give different values for all possible errors in the input. In some cases there are nearly 10 such codes. It seems to me that the CC will ultimately have functions where there are also numerous error possibilities, and where input could not be checked without nearly duplicating the action of the function itself. (I.e., VALIDATE is insufficient.) There should be special registers associated with such functions whose names are made up of the name of the function plus some unusual prefix or suffix. For consistency, current functions should perhaps also switch to this method, rather than returning a special error code as their function-value.

Perhaps there could be a single return code special register, the meaning of whose values would depend on the function last referenced. Since nested functions might be executed, the first part of the special register would be a code indicating the function that last placed a value in it.

7. **AGREE: They suggest a function to return the difference between two dates in terms of the elapsed years, months, and days between them, as well as one that returns the number of days. This is useful for "long interval aging".** (I attach the relevant pages from the book.) (79-85 & 90.)

   I considered suggesting a function to do this, but was defeated by the anomalies inherent in the calculation, which made me unsure of how to define the result. The first two functions on p. 2 of my WP (YR-DAY & YR-MO-DA and their variants) should simply allow "any date format" in argument-3; they would then return date differences (or sums) in YR-DAY and YR-MO-DA format (and their variants).

8. **AGREE: They provide a function that returns true if a date is a holiday or not. The user must fill in a table parameter** indicating which of many possible holidays in the US, Canada, etc. he wants to the function to check for. (The model subroutine they provide includes a sophisticated calculation for the date of Easter.)

   I omitted this function from my list because its table argument requires some unusually "peculiar" information that I didn't possess, and because I felt CC members might object to it because some fuzzy judgment will be required in determining its allowable candidate holidays. (The "where do we stop?" concern.) Overweighing that is the desirability of even an imperfect function of this kind. As the authors write:

   > "As we relinquish more and more control of our destiny to computers, we must humanize them. ... It is time now for the development of some <u>user sympathetic</u> software. Recognition of our holidays and celebrations is but an elementary step in this direction. Like so many broadly accepted notions, this too has an economic basis."

   I suggest the function:

   IF-HOLIDAY (argument-1 argument-2)

   Argument-1 (as always) is any of the three date formats. Argument-2 is the name of a table containing coded entries representing (according to some convention yet to be determined) the holidays that are to be considered as candidates by the function.

   I've spoken to J.T. Murray, and he has indicated his willingness to attend a CC meeting to discuss the seriousness of the problem and his book. (He is Director of MIS at The Rose Packing Co. / 453 Raintree Dr. / Glen Ellyn IL 60137 / 312-381-5700.) The publisher has given me permission to have the attached xeroxes distributed to the CC, and is willing to sell 20-odd copies of the book at wholesale to the CC. (THEY would be sent to a meeting site.)